

ESSAYS ON MANAGING SUCCESSFUL PROJECTS



THE SOFTWARE DEVELOPMENT EDGE

JOE MARASCO

FOREWORD BY MIKE DEVLIN, GENERAL MANAGER, RATIONAL SOFTWARE, IBM

Praise for The Software Development Edge

“This set of articles captures decades of ‘in-the-trenches’ experience across a broad spectrum of software topics. Joe Marasco has the scars and the smarts to articulate patterns of success that can satisfy a broad audience. He uses mathematics, physics, common sense, and storytelling along with a no-candy-coating style to provide unique perspectives on significant problems in delivering software results as a business. Whether you are a computer science theoretician, a frustrated software project manager, a successful businessman, or a skeptical programmer, you will learn a lot from this compilation.”

—Walker Royce, author of Software Project Management:

A Unified Framework (Addison-Wesley), and Vice President, IBM Software Services-Rational

“Joe Marasco’s readable essays on managing successful projects shows that software development managers—no different from all managers—must embrace the fundamentals of management if they are to succeed: working through people and process to be decisive, dealing with politics, keeping on schedule, and, yes, shipping a well-developed product. Marasco uses plain English to explain many integrated skills, ranging from estimating the time it will take to really do things, to negotiating effectively, even to eloquently describing three distinct phases of our personal development. He frequently uses a “can we talk?” conversation with a fictional colleague, Roscoe Leroy, in a Socratic dialogue to illustrate the two sides to a point in many areas (reminiscent of Galileo’s writings to explain his then-heretical views); in this case, Marasco’s advice will help technology professionals escape the clutches of pervasive Dilbertian incompetence, and enable readers to be more effective in our ever-changing world.”

—Carl Selinger, author of Stuff You Don’t Learn in Engineering School:

Skills for Success in the Real World (Wiley-IEEE Press), and Contributing Editor, IEEE Spectrum magazine.

“This is a book loaded with practical experience distilled into insights useful for every software manager, and for thoughtful software engineers. Joe brings his keen observations on engineering, physics, software, and management to bear on managing software projects, and the real-world practices and problem-solving techniques required to be successful.”

—John Lovitt, Senior VP, Rational Software (retired)

“You don’t have to be in the software business to benefit from this book. This is a book that any manager at any level can pick up at any point and enjoy at any time. Highly recommended reading.”

—R. Max Wideman, Fellow, Project Management Institute, AEW Services, Vancouver, Canada

“Marasco’s book makes for fascinating reading for anyone concerned with management problems in general. The reader is introduced to simple quantitative models, based on the author’s experience, that help in assessing performance and choosing a roadmap to successful completion of a complex project. In addition, the author’s use of anecdotes and his writing style make for both an entertaining and informative read.”

—Professor Martin Lesser, Department of Mechanics, Royal Institute of Technology (KTH), Stockholm

“Individually, each chapter is articulate, engaging, thought-provoking, informative, and well informed. Collectively, much as a compilation of short stories from a skilled, mature author, these chapters invite and help train the reader to see into and to understand what is often only observed in passing or scarcely noticed. Seeing the big picture in little things and identifying critical components of the large landscape, Marasco’s analytic and synthetic skills both impress and enlighten.”

—Stephen D. Franklin, University of California, Irvine

“Practical advice on project management expressed in an erudite and entertaining style. The insights are an impressive synthesis of management principles and practical experience that should contribute to improved project management in any organizational activity.”

—Steven Globberman, Kaiser Professor of International Business, Western Washington University

“Joe is an emotionally intelligent leader, and an insightful holistic systems thinker. Both qualities come alive in his writing.”

—Yosi Amram, CEO Coach, and former Founder and CEO, Individual Inc. and Valicert Inc.

“Joe Marasco has assembled a smorgasbord of thought-provoking material that will appeal to anyone who deals with software projects and software developers. Dip into the book at any place and savor new ideas about the nature of software, how to motivate professionals, or one of many other topics that will make you stop and think about your current notions. You may not agree with everything Joe says, but you will respect his position and probably change many of your ideas. Many of the chapters will be required reading for my software engineering students.”

—Gary Pollice, Professor of Practice, Computer Science, Worcester Polytechnic Institute

“A unique and passionate philosophical work, with wisdom distilled from both academic and business careers.... A down-to-earth, subversive, and witty guide for managing the software of machines and of the human soul. Joe shows how the foremost ingredient for success and happiness, in business and in life, is integrity.”

—Kate Jones, President, Kadon Enterprises, Inc. (www.gamepuzzles.com)

“Joe Marasco’s new book is not only a solid introduction in project management discipline, but is also entertaining reading. Using very simple, easy-to-understand-and-relate-to examples, he manages to uncover and describe the most fundamental issues of successful team building and people and project management. His book is a must-read for everyone who wants to get a deeper insight and understanding of this discipline.”

—Boris Lublinsky, Enterprise Architect, CNA Insurance

“There isn’t a manager worth his salt who is going to look for salvation in a ‘how to’ book. What he is going to look for, however, is a nugget or two that will help him out of a specific dilemma or, given time for reflection, provide him with reinforcement that he is on the right track, or trigger thought processes about new ways to solve stubborn, recurrent problems. Well, Marasco provides a mother lode of nuggets for a manager to mine.”

—Bill Irwin, Retired Executive, High Technology Industry

This page intentionally left blank

The Software Development Edge

This page intentionally left blank

The Software Development Edge

Essays on Managing Successful Projects

Joe Marasco



Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U. S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the U. S., please contact:

International Sales
international@pearsoned.com

Visit us on the Web: www.awprofessional.com

Library of Congress Catalog Number: 204118158

Copyright © 2005 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
One Lake Street
Upper Saddle River, NJ 07458

ISBN 0-32-132131-6

Text printed in the United States on recycled paper at R. R. Donnelley in Crawfordsville, Indiana.

First printing, April 2005 (Updated 2/15/11)

To Wini, the light of my life.

This page intentionally left blank

Contents

About the Author	xvii
Foreword	xix
Preface	xxi
 PART 1. GENERAL MANAGEMENT	 1
 Chapter 1. Beginning at the Beginning	 3
The Importance of Good Software	4
Hard Rocks in the Swamp	5
Audience	6
The Iterative Problem-Solving Clock	6
Recap	10
 Chapter 2. Computational Roots	 11
The Precipitator	11
The Answer	12
How This Program Worked	13
Why Was This Generation of Engineers Special?	15
Computation	16
Getting to Know the Numbers by Their First Names	17
So How About Those Computers?	18
Our Computational Heritage	20
Recap	20

Chapter 3. Mountaineering	23
On Climbing Big Mountains	24
Common Causes of Failure	30
Ingredients for Success	31
The Human Factor	32
Recap	32
Chapter 4. Managing	33
Managing Teams	34
Recap	40
PART 2. SOFTWARE DIFFERENCES	41
Chapter 5. The Most Important Thing	43
Iterative Development	43
Roscoe Leroy	44
Going Over the Waterfall	45
The Other Extreme	47
Roscoe's First Picture	47
Roscoe's Second Picture	48
Wait a Minute!	49
Keeping the Vectors Short	49
The Application to Software Development	50
Applied Learning and Short-Vector Direction	51
Risk Targeting	51
Have You Heard This One Before?	52
More on Applied Learning	53
Business Implications	54
The Staffing Effect	55
Just Plain Horse (shoe) Sense	57
Recap	58
Chapter 6. Modeling	59
How to Explain the UML	60
What Is the UML, and Why Is It Important?	61
A Second, Less Trivial Example	61
The Third Example	63
And Now for the Relevance to Software...	65
Raising the Level of Abstraction	66
Recap	66
Chapter 7. Coding	69
How Managers Can Learn a New Programming Language	70
The Problem, Better Defined	71
What Should the Standard Problem Contain?	71
The Animal Game	72
Does the Animal Game Fit the Criteria?	73
Does It Pass the "So What?" Test?	74
It's Your Game	76
Recap	76

Chapter 8. Getting It Out the Door	77
If You Build It, They Will Come	78
In the Beginning, There Was the Sandbox	79
Why Should the Product Build Be Hard, Anyway?	79
What About Iterative Development?	84
Recap	85

PART 3. THE PROJECT-MANAGEMENT VIEW **87**

Chapter 9. Trade-Offs	89
The Project Pyramid	90
Five, Not Four	92
Enter the Pyramid	93
The Altitude Variable	94
The Pyramid's Volume Is Constant	94
A Statistical Interlude	95
Right Idea, Wrong Distribution	97
Implications for Real Projects	99
What Does It Take to Get to a Coin Flip?	99
More Confidence	100
Important Caveats	101
It's All About Risk	105
Recap	106

Chapter 10. Estimating	107
What If We Used Common Sense?	108
Chocolate Versus Vanilla	108
Roscoe Explains	109
Roscoe Goes Deeper	109
Roscoe's Calendar	110
Roscoe Computes	111
Roscoe Gets into Software	111
Roscoe Reports In	111
Guess We Did Something Right	112
Roscoe Sums It Up	113
Roscoe Picks a Bone	113
Guess We Did Something Right, Part Two	114
Roscoe Admitted to Software Project Manager Fraternity	115
Recap	115

Chapter 11. Scheduling	117
Roscoe Poses the Problem: How Late Are You Gonna Be?	118
Joe Makes a Slight Comeback	119
Roscoe Returns	120
Roscoe's Rogue's Gallery	120
Roscoe's Graph	121
One Last Objection	122
Roscoe's Parting Shot	123
Recap	123

Chapter 12. Rhythm	125
A Physicist Looks at Project Progress	126
Reality Intrudes	134
What About Iterative Development?	135
One Last Graph	140
Recap	142
 PART 4. THE HUMAN ELEMENT	 143
 Chapter 13. Politics	 145
Context	146
Definition	147
Three Scenarios	147
Politics Is Inevitable, But...	148
When Things Get Political	149
The Engineering Mapping	152
High-Trust Environments	153
Other Variants of Bad Politics	154
Recap	155
 Chapter 14. Negotiating	 157
Communication Is Everything	158
Roscoe Explains His Theory	158
Are We Done Yet?	164
Recap	165
 Chapter 15. Signing Up	 167
Roscoe Gets His Nose Bloodied...	168
...And Immediately Cuts to the Chase	168
Vesuvius Erupts	168
How They Do It in Texas	169
The Relevance to Software	170
The Dog Ate My Homework	170
Spec Wars?	171
The Three Most Common Excuses	172
And Another Thing...	174
Thrust, Parry, and Riposte	174
Large Project Chicken	175
The End of Software Development As We Know It?	175
Elaboration Versus Construction	176
Tough Love	177
Recap	177
 Chapter 16. Compensation	 179
Going for the Flow	180
Flow and Software Development Performance	181
Applying the Flow Model to Compensation	182
Money Isn't Always the Answer	191
Recap	192

PART 5. THINKING Laterally	195
Chapter 17. History Lesson	197
Don't Let the King Be Your Architect	198
Things Aren't Always As They Seem	198
Checking the Design	198
Knowing What You Don't Know	198
Continuity of Leadership	199
In a Hurry, As Usual	199
Focusing on the Wrong Features	199
When the Design Is Bad...	199
The Relevance of Testing	200
Prototype Versus Product	200
The Inquest	200
Recap	200
Chapter 18. Bad Analogies	201
Houston, We Have a Problem	201
Fig Newtons	203
Everything's Relative	205
Quantum Nonsense	207
Heat Death	211
Other Examples	213
Good Science	213
Recap	214
Chapter 19. The Refresh Problem	215
Refreshing Embedded Software	216
The Current Situation	217
The Software Upgrade Game	218
A Modest Proposal	218
Software Upgrades, Revisited	219
Some Nice Things Come for Free	220
Why This Will Work	221
Refinement	222
What About Software Piracy?	223
Until the Sun Takes Over	223
Recap	224
Chapter 20. Not So Random Numbers	225
Roscoe Sets the Stage	226
Simulating the Batter	226
First Steps	228
Second Steps	230
Generating More Probabilities	230
Of Course, We've Already Left the World of Baseball	232
Reality Is Ugly	233
Monday's Solution	234
Lessons Learned	239
Recap	240

PART 6. ADVANCED TOPICS	241
Chapter 21. Crisis	243
The Five Days of the Fish	244
The Fish Market	244
Day 1: Unaware	244
Day 2: Avoiding the Issue	244
Day 3: Enter “The Fixer”	245
Day 4: The Turning Point	245
Day 5: Two Critical Paths	246
Moral of the Story	246
Recap	247
Chapter 22. Growth	249
Growth Issues	249
The Naïve Model	251
Consequences of the Model	253
An Illustrative Example	259
Non-Linearity	260
Call to Action	261
Conclusions	263
Nomograph	264
Spreadsheet	265
Recap	267
Chapter 23. Culture	269
What Is a Culture?	270
Strong and Weak Cultures	271
Defining Corporate Values	271
And the Applicability to Software Is...	274
Building a Strong Culture	275
When You’re Looking for a Job...	279
The Bottom Line	280
Recap	280
Chapter 24. Putting It All Together	281
Schlepper	282
Macher	284
Mensch	286
More on Mensches	287
Population Distribution	288
Some Final Thoughts on the Model	289
Recap	290
Acknowledgments	293
Index	297

About the Author

Joe Marasco is a retired senior vice-president and business-unit manager for Rational Software, now one of the five brands of the IBM Corporation. He held numerous positions of responsibility in product development, marketing, and the field sales organization, overseeing initiatives for the Rational Apex product and Visual Modeler for Microsoft Visual Studio. In 1998, he served as Senior VP of Operations. He retired from Rational in 2003. He holds a bachelor's degree in chemical engineering from The Cooper Union, a Ph.D. in physics from the University of Geneva, Switzerland, and an M.S.A. from the University of California, Irvine Graduate School of Management. When not writing, he barbecues and plays golf; his ribs are much tastier than his scores.

This page intentionally left blank

Foreword

Why should anyone listen to what Joe Marasco has to say about software development and the people who do it?

In the spring of 1991, we awarded Joe his five-year service award. Back then, Rational Software was small enough to individualize these awards, and Joe's was quite unusual. We obtained the hood ornament from a Mack truck—the classic bulldog—and mounted it on a plaque. Everyone agreed that this award symbolized Joe's dedication and tenacity when it came to getting the job done.

So it wasn't too surprising when, later that year, we selected Joe to lead a watershed development effort at Rational. At the time, our flagship product, the Rational Environment, ran on proprietary hardware, and we realized the importance of moving it to the UNIX platform. While this move was inevitable, it was fraught with risk; in fact, many other companies suffered fatal damage trying to move their software solution from proprietary hardware to industry standard platforms—among the carcasses littering the side of the road were electronic design automation (EDA) companies such as Daisy and language-specific companies such as Symbolics. The task was known to be difficult, the result uncertain, and the need essential.

Joe, on the other hand, was positively ancient by Silicon Valley standards: he was 46. But we believed that Joe's experience and "steady hand at the tiller" would see this project through. We also knew that Joe would do whatever it took to get the project done.

The record speaks for itself. In September 1991, Joe took on the leadership of the new team with a plan to deliver “Rational II” in two years on two UNIX platforms. After seven months, a limited-function subset prototype was up and running. After 16 months, the development team was “self-hosted,” which meant that it was able to complete the development of the product using the partially-completed product itself. And, to the minute, the team delivered what became known as Rational Apex on two UNIX platforms—IBM and Sun—in the two years that had been promised.

Apex was an extremely successful product, one that is still delivering value to customers today. Joe was the Business Unit Manager through releases 2.0 and 3.0 and also oversaw its porting to every significant UNIX platform and to the Windows platform as well. More important, for the 10 years following the release of Apex 1.0 in 1993, Joe was the “go-to” guy whenever we had a difficult product delivery problem in the company. As the company grew through merger and acquisition, Joe assumed the role of troubleshooter, helping out wherever the need was greatest, wherever the pressure was most intense.

One of the reasons Joe was so successful in delivering products is that he spent a lot of time with his developers understanding the details of the products and the development problems. But he also spent a lot of time with Rational’s customers, developing a keen understanding of their needs. As every product delivery is the result of many compromises, Joe was always well informed to make good judgments when it came to product decisions.

Near the end of his career at Rational, Joe began to write about software development in a series of articles in Rational’s e-zine, *The Rational Edge*. Unlike the articles written by our “Three Amigos,” these were much more down in the trenches, reflecting his experiences both at Rational and in his previous environments. What we discovered was that Joe was able to cogently articulate his experience and act as a “virtual mentor” for budding software development managers worldwide. The response to these articles was extremely positive, and it is a pleasure to see them all collected here in one place.

This is not a theoretical treatise on software development. That’s not what Joe was all about. Joe was about having his teams ship products we could all be proud of, products that were easy to maintain and offered real value to customers. If you want to develop products that you can be proud of, that lend value to your customers, this book is a “must-read.”

Mike Devlin
General Manager
Rational software, IBM

Preface

This book draws heavily on a series of columns called Franklin’s Kite that appeared in Rational Software’s e-zine, *The Rational Edge*, in the early 2000’s. These articles were aimed at software development managers, and their goal was to help readers avoid many of the common pitfalls that await software development projects and teams. More than 20 appeared, and we—my editor, Mike Perrow, and I—noticed that readers often began their monthly perusal of *The Edge* with *The Kite*.

My intention here is to not only to collect these articles but to sew them together in a form that makes them even more useful for software development managers *and their managers*. I have done that by reorganizing them thematically, instead of presenting them in the order they originally appeared. This has caused me to do some light editing in places where “forward-referencing” would otherwise take place. I have also paid attention to the footnotes, many of which appeared in the original as URLs and appear here as more formal citations where appropriate. Finally, I have added material at the beginning and end of each chapter so that the context of each article as part of the whole becomes clearer.

The reader will quickly note that the chapters have several different styles. Some of them are expository, some are fairly analytical, and some are folksy “Socratic dialogs” between the author and his avatar, one Roscoe Leroy. Roscoe is an invented character, a

good technical general manager who initially knows little about software development. I use him as a foil, allowing his “naïveté” to force me to explain things without using technical jargon. My approach is ecumenical and subversive: I will use any technique that permits me to get the message across. Some of these chapters will appeal to some readers, and others will appeal to others. Whatever works is, by definition, good. I take my cue from Horace, who wrote in *The Art of Poetry*, “He has won every vote who has blended profit and pleasure, at once delighting and instructing the reader.”¹

I divide the work into six parts of four chapters each. Briefly:

- **General Management:** These chapters deal with topics that are useful to managers in general and also expose the reader to my background and biases. I include them so that we have a common baseline for what follows.
- **Software Differences:** In this section, we take a look at those things that distinguish software development from other management challenges.
- **The Project-Management View:** I take the perspective that a software-development project is a variant of the generic project and, as such, amenable to classical project-management techniques. On the other hand, I strive to point out what *is* different about software development.
- **The Human Element:** I turn around in this section and look at software development from the perspective of the people who do it. Once again, I try to compare and contrast that which is similar to that which is different for software-development projects.
- **Thinking Laterally:** Software people come at problems from many different points of view. In this section, I expose the reader to some of the more speculative and original ideas that he or she may not have seen before.
- **Advanced Topics:** The successful software-development manager is like a really good pinball player: His reward for high scoring is given in free games. This additional “stick time” leads to his becoming even more proficient. In this section, I talk about some of the challenges that come with success.

This book has 24 chapters.² You can read it serially, or pick out a chapter at a time; they can stand on their own. This is a good “airplane book”; read a chapter and then think about it for the rest of the flight. If you get just one new idea from one of the chapters that covers the price of the book, I will have been successful.

With these prefatory remarks out of the way, let’s get down to it.

¹ Horace, *Satires, Epistles, and Ars Poetica* (Cambridge, Massachusetts: Harvard University Press, 1999). The original Latin text is “Omne tulit punctum qui miscuit utile dulci, lectorem delectando pariterque monenendo.” It can be found at line 343 of *Ars Poetica*.

² Coincidentally, so does *The Iliad*.

CHAPTER 8

Getting It Out the Door

I have on occasion claimed that I can build the perfect product. Just don't ask me to ever ship it.

As soon as you require that I ship a product on a given date, I can guarantee you that the product will be imperfect. It will disappoint someone along some dimension. It will lack some feature, exhibit some annoying minor bug, or will lack some piece of documentation. No doubt there will be rough edges in its user interface. If only we had more time...

This is not a phenomenon unique to software products. A shipped product is always a compromise between the product we would ideally like to ship—the one that approaches perfection—and the one we really need to ship because we must begin generating revenue. And sometimes, believe it or not, the product we ship *is* good enough, even though it represents a compromise. The test is whether or not it serves the greatest good for the greatest number.

Consider an *update release* of an existing product, one that will add some new features and fix many annoying defects resulting from the previous compromised release. You can work on this update as long as you like; the longer you take, the more features you can add and the more bugs you can fix. But here's another way to look at it: The longer you wait to ship that update release, the longer your existing customers will have

to live with the bugs in the version they are currently using. So the tradeoff becomes this: Is it better to ship 50 bug fixes today, or 55 in another two weeks? If you have thousands of customers who are suffering with Bug #29 on the list every day, I think I can make a pretty good argument for shipping *yesterday*.

Once you realize that shipping the product is not only part of your job but in fact the critical step—Bob Bond¹ would call it “running it through the tape”—you need to consider exactly what is required to go from some assemblage of working bits to a package that you can put on the loading dock or, alternatively, some set of files that you can stage on your download server. You need to consider testing, installation, documentation, preparing the support organization, and many, many other details. Like the death from a thousand cuts, getting this all right can be extremely painful the first hundred or so times you do it. It is one of those exercises that require method and persistence, and extremely meticulous follow-up.

The purpose of this chapter is not to bludgeon you to death with the obvious. What I focus on in this chapter is a small subset of the problem: How do we “close out” development of the software so that we can ship the product? When we are on “final approach” to shipping the product, what changes? The answer is this: If you have been doing it right, the change is imperceptible. If you have neglected thinking about this problem all along, then you will suffer large, severe, and disruptive change at the end, and your ability to ship will be endangered.

If You Build It, They Will Come²

In the world of software products, there are successes and failures, determined by the free market system. We must, of course, add to the list of failures those projects whose products never see the light of day—the ones that are worked on for various lengths of time but never ship. As obvious as it sounds, you cannot be successful unless you meet the precursor of shipping your product.

As you cannot ship what you cannot build, actually putting together the pieces becomes critical. Intrinsic in this is the concept of a *repeatable* build process. You will build the product over and over again, until one of your *candidate releases* passes muster and you let it out the door. I now confront the issues involved in creating such a repeatable build process for your product.

¹ Bob Bond ran sales and marketing at Rational for many years. He was a very positive mentor for me.

² In fact, the line from the movie “Field of Dreams” was, “If you build it, he will come,” the “he” being either Shoeless Joe Jackson or the principal character’s father. The line has been so frequently misquoted that most people use this one. Of course, at the end of the movie “they” come, as illustrated by the stream of headlights across the Midwestern plain.

In the Beginning, There Was the Sandbox

Products come out of projects, and projects tend to begin in haphazard ways. Organizations with well-defined processes have developers building their components in local work areas, sometimes called *sandboxes*. They provide for mechanisms whereby the sub-products of these sandboxes can be assembled, sometimes in *ad hoc* ways, so that each development team can test its progress in the context of the whole product. Configuration management systems allow for appropriate partitioning such that each developer (or team of developers) has the autonomy and isolation to work on his piece without stepping on the other guy's toes, while at the same time providing for a loose integration context.

This works fine in the early, chaotic days when everything is changing very rapidly, and before architectures are well-defined and interfaces are nailed down. However, before too long even modest projects outgrow this framework. At that point, one of two things happens: Either the organization makes *the build* a priority and adds some structure, or it doesn't. In general, those that do establish a regular “heartbeat” for the project—a periodic, regular, and dependable build cycle—improve their chances for success. Those that don't establish this rhythm find that entropy begins to take over, and that building the product becomes more difficult over time.³

Many organizations vastly underestimate the effort it takes to put a good build process in place. Because of this, projects in their latter stages often have a “new” problem to deal with: In addition to having buggy software, incomplete parts, and so on, they also struggle with something that they have taken for granted—the simple assembly of their product. This is a trap for the unwary. In order to not fall into the trap, you need to understand more about the process of assembling a product.

Why Should the Product Build Be Hard, Anyway?

First of all, the product you are going to ship has more pieces to it than the prototypes you have been putting together for internal consumption. Here's a classic example: Developers and testers rarely look at the help system, because they know the product well enough to play with it and test it. Once you are going to have outsiders try to use it, you need a well-elaborated and working help system for people to use. Further, you need instructions for installing the software in different computing environments, as well as various other adjuncts that you can live without when you are only consuming your software

³ *Entropy* is the tendency that all systems have to move from an orderly state to a disordered state when left alone. It is a fundamental physical law. One might say that all attempts at progress, by any civilization, fly in the face of entropy. Another way to say this is that to bring order out of chaos takes work, and that once you stop working, entropy will cause the system to spontaneously move to a more disordered state. I will talk some more about this in Chapter 18, “Bad Analogies.”

internally. So the first problem that comes up is one that might be dismissed as *packaging*. You need more pieces to ship a product than to use it internally; and further, you need to document all the little details that the internal team has always known or taken for granted.⁴ Making the product ready for outside consumers is sometimes called *sanding off the rough edges*. Some of these “rough edges” can be very sharp, and because you don’t catch them all the first time, your first consumers may cut their fingers on them.

Let’s assume, however, that this is just a logistical exercise and that with enough planning you can avoid the “packaging” trap. In some sense, it can be put in the “annoying detail” category: If you ignore it, it will bite you; but if you are aware of it and plan for it, then it is relatively easy to overcome. So, forewarned is forearmed: Treat packaging as a purely technical problem and you will be fine.

In fact, there are three much more fundamental obstacles to success that come up over and over again. They are distinct and interrelated, and all three must be worked on to achieve a successful build process.

Obstacle 1: Organizational Politics

Many software development managers lose sight of the simple fact that controlling the build process is first and foremost a political problem. To put it simply, he who controls the build has an enormous amount of power. After all, the build cycle itself defines the rhythm of the entire development and test organization. Think of the build cycle as the software equivalent of a factory assembly line. The person who gets to define the characteristics of the line and its speed determines, to a very real extent, the output of the factory. Line workers are very aware of their subservience to the line. The cardinal sin in the factory is to slow down, or—Heaven forbid!—shut down the line. The software equivalent is submitting a set of changes that *breaks the build*.⁵

Now the build process is something that everyone must participate in, but only one group can control. By its very nature it is not a democratic enterprise; it requires a certain amount of hierarchical and structural apparatus to work at all. Everyone agrees on this, more or less. The sticky wicket is determining who gets the responsibility and authority to make it work. That group will, from that day forward, wield a lot of power and clout.

Because human beings are, in general, reluctant to give up this sort of power, the build process becomes a political football. Myriad discussions ensue as to who will have

⁴ The standard vehicle for this is called the *release note*. The release note documents the limitations of this version of the software, known bugs, and so on. It is an attempt to characterize the state of the deliverable, as it is better to tell your consumers things you know about rather than have them discover them on their own. Sometimes the release note is called the *readme file*.

⁵ There are legitimate reasons for shutting down the line, and sometimes the person on the factory floor is the most appropriate person to do this. On the other hand, shutting down the line by mistake is definitely not a good idea.

the right to do what to whom in the interest of the build process. All of the negative political tendencies of your organization will be exposed during these discussions.

The purists among you will cry out that political tendencies should be discouraged or even condemned, pointing out that the job is hard enough from a technical point of view and should not be “polluted” by politics. In most organizations, however, *wishing* politics away will not necessarily make them *go* away. Politics is a fact of life that must be dealt with.⁶ However, you must get through this phase, as unpleasant as it first appears. Else, you will be incapable of dealing with the next two hurdles.

Here are some specific suggestions:

- Try to get the group to agree that someone has to be in charge, because a loose confederation approach is doomed to failure.
- Try to reach a reasonable compromise between the autonomy of the constituent teams and the centralized authority that will be required.
- Always make sure that the management team understands the importance of the issue and has the very best people assigned to the build.
- Later in this chapter, we will talk about having a *czar of the build*. Make sure it is a person who is technically competent, firm, fair, and respected by everyone. Install him or her early in the process and have this person guide you through the political shoals.
- Enlist management’s support in crushing “bad politics,” should it rear its ugly head.

Obstacle 2: The Process

Having hacked through all the political jungles that accompany conceding power to the build group, the participants must now agree on the process they will use. Just as form follows function, the process will often be shaped to mirror the political compromises that were made to get to this juncture. There is plenty of interaction between the first and second obstacles. In fact, often the process obstacle presents itself early on, in Phase 1, because it is being used as a surrogate by those who don’t want to openly admit that there are unresolved political issues. In some organizations, we see these two obstacles mashed together into one giant hairball, which in turn gives “process” a bad name. You cannot use “process” to solve what are intrinsically political problems, much in the same way that you cannot “solve” technical problems through political compromise.

⁶ My perspective is that there are “good politics,” akin to the notion of “fighting fair,” and that a healthy political process can and should work toward making good decisions. Then there are “bad politics,” which make organizational objectives subservient to personal agendas and self-aggrandizement; this sort of politics needs to be stamped out wherever it is found. The problem, of course, is the gray zone in between. I treat this subject in more detail in Chapter 13, “Politics.”

The basic tension at this point revolves around the people who want a strict, rigorous process—sometimes called *lots of rules and no mercy*⁷—versus the people who want a looser set of policies. Acknowledging that there is no single, simple, right answer is usually the best place to start here. Your process will have to be tuned to your organization, because all organizations have their peculiarities.⁸

That does not mean that you need to invent new process. I used the word “tune” advisedly in the previous paragraph because I am firmly convinced that the best way to deal with this issue is to start with a base process that has been demonstrated to work before. Rational Software’s Unified Change Management (UCM), for example, has a rich legacy of successful application. We know it works across a broad spectrum of domains, applications, and organizations. Why start over? Do you really think you are going to do better?

There are a few traps you don’t want to fall into at this point. One is the *religious wars* pitfall. In every organization there are process gurus who believe that they, and only they, have the magic formula. And sure enough, every time there are others who resist, quite certain of their own convictions.⁹ Regardless of who is right or wrong, these crusades are totally unproductive, often revolving around obscure details of little importance. The strong manager needs to identify the religious process fanatics and stifle them early. Sometimes the only answer is to tell them to put a cork in it. Remember always that process is not an end in and of itself; it is a means to an end—shipping the product!¹⁰

Another trap is to think that any process, no matter how good, can substitute for thought or judgment. For every ironclad rule, there is bound to be an exception. You will have to watch what is going on and make midcourse corrections, no matter what your

⁷ I believe the world is indebted to James E. Archer for this characterization. Jim is one of the most effective development managers I know, having been the godfather for Rational’s programming environments products from the very beginning. He and I had many interesting discussions on the right amount of process.

⁸ Some people argue at this point that you should endeavor to get your process “right” and then tune your organization to fit the process. While this is a laudable objective and *theoretically* the right approach, I have rarely found it to be successful in practice. You cannot allow a regressive organization the prerogative of rejecting reasonable process; on the other hand, it is difficult to implement any process that is too far out in front of the organization that must carry it off.

⁹ To illustrate how far out of control this can become, the wars are often characterized as struggles between the “process Nazis” and the “anarchists.” With such value-laden labels, it is difficult to have discussions that will get to the right place.

¹⁰ In a like manner, the anarchists will be hard put to demonstrate that they can ship product without any process. As is the case in almost all these debates, neither extreme position is defensible.

process is. As called out previously, you will need to modify and tune your process in real time as you discover what works for you and what doesn't.

Lastly, get on with it. Perfect is the enemy of good.¹¹ You will develop your process iteratively, just the way you develop the software. Get to Iteration 1 quickly. Learn. Change. Improve. Repeat until done.

Obstacle 3: Tools

Just as the first obstacle (politics) and the second obstacle (process) are intimately related, so are the second and the third. The third, of course, is the toolset that you will use to implement the process. Needless to say, choosing the tools first is getting it bass-ackwards, but surprisingly enough, that's the way many organizations go about it. They then wind up with the tool determining the process, which can be loads of fun when the process thus derived is inconsistent with the political philosophy of the organization.

Obviously, you need tools that will automate and enforce the process you have chosen to use. If you have a process that admits mistakes, you will be "backing out" changes from time to time. Does the tool support that easily? Are developers going to be checking in their work to a common baseline from multiple remote sites? If so, then your tool had better support that model. Do you want to build your entire product from top to bottom every night? If so, then I hope your tool has the performance and turnaround characteristics that will permit that. Do you want to automate your regression testing as part of the build? Once again, tool support is crucial.

Even organizations that have done a good job with the first two problems sometimes flounder with the third. And sometimes it is not the tools' fault, either. Once again, using our factory analogy, you need someone to monitor the line and to do quality control for the product coming off the line. Without constant vigilance, it is easy to automate a process that produces a low-quality result. Every successful build process requires a foreman or the equivalent thereof; sometimes he or she is called the *czar* (or *czarina*) of the build or, more simply, the *buildmeister*.¹² The buildmeister monitors the health of the line and makes sure that a steady stream of good-quality product is produced.

¹¹ I first heard this from Mikhail Drabkin of Riga, Latvia, and assumed it was Russian folk wisdom. He may have in fact been (mis?)quoting Soviet Admiral Sergei Georgievich Gorshkov, although that citation turns out to be only an approximation: Gorshkov's ended with "Good enough." Dr. Stephen Franklin of U.C. Irvine points out that very similar sentiments have been attributed to both Clauswitz and Voltaire. Clauswitz is much more verbose, and there seems to be evidence that Voltaire "borrowed" from an Italian proverb. Though these pithy sayings sometimes have a provenance that is difficult to pin down, one can deny neither their truth nor their wisdom.

¹² Here's a cautionary, funny, and politically incorrect tale: I once made a big deal about having a czar of the build, and then appointed a fellow who was, shall we say, altitudinally challenged. He unfortunately became known as the czardine of the build. Ouch!

One last semi-technical note: Beware of the old saying, “We can always write a script that can do that.” The problem is that these scripts always start out small and simple and then grow in ways that are random and unsupervised. Scripts, unlike programs, are rarely designed; they just grow. They become compendia of special cases and are inadequate to respond to the ever-increasing demands of the organization; they are brittle. They are a maintenance nightmare, especially if the original author moves on. And they are very, very difficult to debug. Just as the road to Hell is paved with good intentions, the road to “build Hell” is paved with the out-of-control products of general-purpose scripting languages.¹³

What About Iterative Development?

Iterative development sidesteps one of the great dangers of the waterfall approach: leaving system integration to the last minute. One of the reasons so many waterfall projects fail is that, very late in the game, developers are trying to assemble their product for the very first time. In addition to finding many bugs, mostly in the interfaces, they grapple with the normal logistical and organizational problems of putting together a build chain for the first time. Often, things that pass for bugs are nothing more than the artifacts of broken builds. But the organization is in such chaos at this point—running out of time, nothing working, people frazzled—that it is hard to separate the sugar from the salt. It is also a very bad time to be trying to solve political and process problems.

By contrast, iterative development requires that you construct your build chain to accomplish the deliverable for Iteration 1—a working program. So you begin to debug this process early in the project, not at the end. By the time you get to Iteration 3 or 4, the build process is actually starting to work pretty well. For the last iteration, the one that will deliver the final bits, the build should be working like a finely lubricated Swiss watch.

As with pretty much everything else in software development, there are a small number of ways to get this right and almost an infinite number of ways to get it wrong. If you view the build as a detail that will “just happen,” then the odds are against you. Make sure you attack the build process as a conscious effort that is critical to your success, and devote the time, energy, and resources to it that it demands. To do any less is sheer folly.

¹³ Some scripting languages have been favorably likened to duct tape; would you want to begin the final assembly of your jet aircraft by having some guy yell out, “Time to get out the duct tape!”?

Recap

I've frequently been called in late in the software development cycle when projects are in trouble. Often, it is difficult at first to judge the depth of the yogurt. Usually the developers are focusing on how much they are "behind," as measured in things that are coded but just don't work, and things that are not coded at all but should be.

While this is one important aspect, I always begin to also look at the health of the build process. If the build process is non-existent or badly broken, it needs attention *immediately*. The reason for this should be obvious: at this stage of the proceedings, the lack of a reliable, repeatable build process will impede all further progress. You can't test what you can't build, and repeated testing is a necessity at this point; else, how can the developers know what they've fixed and what remains problematic?

Sadly, in many organizations the build process is something that is relegated to the B Team. This is a huge mistake. You must have A-list players in this part of the organization. As soon as people understand how much you as the senior manager appreciate the contribution of this team, you will have no problems getting volunteers.

The other thing that proves very slippery is answering the question, "How do we know when we are done?" Getting agreement well in advance on some clear criteria is incredibly useful; such agreement reduces the odds that the bar will move radically up or down as the ship date looms. One of the major objectives when moving into the Transition Phase of iterative development is to define reasonable criteria for shipment. Without a clear "exit plan," the project risks a series of never-ending last-minute slips.

Thus ends Part 2 on the basics. In Part 3, I will look at software development from a project-management perspective.

This page intentionally left blank

Index

A

- abstraction, 66
- acceleration, 127
 - of a vertical ball toss, 129-130
- accidental degeneracy, 231
- accuracy, 109
- achieving flow, 181
- acquisitions, 277-278
- adding
 - employees, 250
 - useful hours to product, consequences of naïve model, 256
- algorithms, quadratic algorithms, 172
- altitude variable, 94
- American-born engineering students, 15
- analogies, 202
 - Newton's Laws of Motion, 203-205
 - oil changes, 217
 - physics, 201-203
 - quantum mechanics, 207-211
 - relativity, 205-207
 - thermodynamics, 211-212
- anarchy, 47
- "The Animal Game," 72-74
 - So What? test, 74-75
- anticipating obstacles, managing teams, 35-36
- applications, mission-critical applications, 4
- applied learning
 - iterative development, 53-54
 - short vectors, 51
- Archer, James E., 82
- Arrow, Kenneth, 212
- assumptions, naïve model, 252-253
- attrition, 267
- audience for this book, 6
- avoiding issues, 244

B

bad science, 213
 Barry, Dave, 17
 bartering, 149, 151
 baseball, fantasy baseball, 226-228, 230, 232-235, 239
 batch processing, 19
 batteries, 218
 solar power, 223
 batterychip, 220-221, 223
 "Being in the zone," 180
 Bernstein, Dave, 55
 Bilenko, Herman, 17
 Boehm, Barry, 113
 Bond, Bob, 78
 Booch, Grady, 137
 Borgenstam, Curt, 198
 Brooklyn Bridge, 199
 Brooks, Frederick P., 102, 118, 250
 Brownian Motion, 47
 build processes, 79
 difficulty of, 79-82
 organizational politics, 80-81
 tools, 83-84
 iterative development, 84
 build rhythm, 125
 buildmeister, 83
 business implications, waterfall development versus interactive development, 54-55

C

calculating, 225-226
 Leroy, Roscoe; adventures while
 being shipwrecked, 226-228, 230, 232-235, 239
 calculators, 18
 calibrating schedules, 120, 122-123
 candidate releases, 78
 cell phones, 217

Central Limit Theorem, 104
 Cha, Sandra E., 269
 Chaos Theory, 213
 Churchill, 284
 citizenship, 272
 Clauuswitz, 83
 closure, communicating with engineers, 162
 code, counting lines of, 114
 code reviews, 59
 code rot, 216
 coding, 69
 cognitive dissonance, 102
 commitment, 168, 174-175
 deadlines, 171-172
 definition of, 170
 elaboration and construction, 176-177
 excuses, 170-173
 high-trust environments, 175
 honoring, 174
 large project chicken, 175
 scheduling, 175-176
 Texas handshake, 169-170
 communication, 158-159, 162-164
 closure, 162
 establishing ownership, 160-161
 iteration, 164-165
 not suggesting solutions, 161-162
 pseudoscientific jargon, 213
 remember who you're dealing with, 159-160
 compartmentalization effect, 108
 compensation, 179, 191-192, 276
 flow, 182
 Cone of Correct Compensation, 185-191
 diagonal cases, 186-187
 job-based models, 184-185
 skills-based models, 183-184
 productivity/performance, 181-182
 win-win, 190-191
 competition, 29

- completion, 138
 - compromise, 148-149
 - computations, 16-18
 - computers, engineers from 1960-1970, 18-19
 - Cone of Correct Compensation, 185-191
 - mapping team members, 191
 - Conestoga, PA, 50
 - consequences of naïve model, 253-255, 257-258
 - adding useful hours to product, 256
 - cost, 257-259
 - multipliers, 254-256
 - constants in the software development business, 5-6
 - constraints, 161
 - construction, 137, 139-140
 - commitment, 176-177
 - continuity
 - culture, 276
 - growth, 276-277
 - mergers and acquisitions, 277-278
 - new efforts, 278-279
 - single big customers or partners, 278
 - of leadership, 199
 - contribution of new hires during transition, 251-252
 - Cool Hand Luke*, 119
 - Cooper Union, 15
 - core values, 272
 - corner cases, 227
 - corporate values, defining, 271-272
 - customer focus, 273
 - integrity, 272-273
 - results, 273-274
 - correct compensation, 183
 - cost
 - consequences of naïve model, 257-259
 - labor cost, 260
 - counting lines of code, 114
 - crisis, 243
 - avoiding issues, 244
 - critical paths, 246
 - dealing with, 247-248
 - fish, 244-246
 - fish markets, 244
 - fixing problems, 245
 - turning points, 245-246
 - crossing the chasm, 131
 - Cskiszenhalmihalyi, Mihaly, 180, 288
 - cultural differences, politics, 146
 - culture, 270, 280
 - continuity, 276
 - growth, 276-277
 - mergers and acquisitions, 277-278
 - new efforts, 278-279
 - single big customers or partners, 278
 - job hunting, 279-280
 - leadership by examples, 276
 - strong cultures, 271, 280
 - weak cultures, 271
 - curves. *See also* graphs
 - completion curves, 138, 140
 - force curves, 133-134
 - human behavior, 130-131
 - learning curves, 130, 138, 140
 - percent completion curves, 126
 - project velocity, 132
 - and reality, 134-135
 - S-curves, 130
 - customer focus, 273
 - customers, 276
 - continuity, 278
 - czar of the build, 83
- ## D
- Dangerfield, Rodney, 158
 - Dante, 276
 - Darwin, Charles; Theory of Evolution, 213

Davis, Derrick, 91
 deadlines, commitment, 171-172
 debuggers, 210
 decisiveness, 28
 derivatives, 127
 design, Vasa, 199
 devices, 217
 buying batteries packaged with
 software, 218-223
 cell phones, 217
 upgrading, 217
 diagonal cases, flow and compensation,
 186-187
 dialects, 60
 dice, 226
 distributing software development
 projects, 96
 distribution, project pyramid, 97-99
 documentation, 8-9
Dr. Dobb's Journal, 126
 Drabkin, Mikhail, 83
 drafting, 15
 drag, effect of new hires on existing
 team, 252

E

Edmonson, Amy C., 269
 education, 14
 educational system, 12
 Einstein, Albert
 relativity, analogies, 205-207
 Special Theory of Relativity, 207
 Theory of Relativity, 207
 elaboration, 137, 139-140
 commitment, 176-177
 electromagnetism, 211
 embedded software, refreshing, 216-217
 empathizing, 7-9
 empire building, 155
 employees
 adding, 250

 attrition, 267
 new hires, 250
 contributions during transition,
 251-252
 effect on existing team, 252
 evaluating, 262
 engineering, 13
 engineering discipline, 4
 engineering mapping, 153
 engineers, 153
 1960-1970, 15-16
 computers, 18-19
 heritage of, 20
 politics, 152-153
 entropy, 79, 211
 error rates, 215
 establishing ownership, 160-161
 estimating, 107-111, 113-114, 119-120
 schedules, commitment, 175-176
 estimation, 17
 eternal verities, 5
 Euclid, 112
 evaluating new hires, 262
 evolution, 213
 examples
 leadership, 276
 of naïve model, 259-260
 of politics, 147-148
 excuses for breaking commitments, 172-173
 experience, selecting teams, 25

F

failure, common causes of, 30-31
 false-positives, 3
 fantasy baseball, 226-228, 230,
 232-235, 239
 Faraday, Michael, 211
 "Father Knows Best," 13
 featuritis, 199
 feedback, 9
 Feynman, 18

“Field of Dreams,” 78
 finite horizon, 97
 first-level disconnect, 69
 fish
 crisis, 244
 death of, 244-246
 moral of story, 246
 fish markets, 244
 fixing problems, 245
 flow
 achieving, 181
 compensation, 182
 Cone of Correct Compensation,
 185-191
 diagonal cases, 186-187
 job-based models, 184-185
 skills-based models, 183-184
 productivity/performance, 181-182
 flow channels, 180-181, 288
 focus, 29
 managing teams, 34, 37
 forces curves, 133-134
 FORTH, 75
 FORTRAN, 19
 Franklin, Dr. Stephen, 83
 frugality, 93

G

Galileo, 127, 206
 generating probabilities, 230
 GIGO (Garbage In, Garbage Out), 19
 goals, 27-29. *See also* milestones
 Godel’s Incompleteness Theorem, 213
 Godfather offers, 37
 gonifs, 283
 good science, 213
 Gorshkov, Sergei Georgievich, 83
 Gould, Stephen Jay, 213
 Grady, Booch, 113
 graphics, 59-60

graphs. *See also* curves
 parabolic position graphs, 130
 project completion graphs, 130
 total forces on a project, 141-142
 Gresham’s Law, 277
 Gresham, Sir Thomas, 277
 growth, 249
 attrition, 267
 continuity, 276-277
 employees, 249-251
 mergers and acquisitions, 277
 naïve model, 251
 assumptions, 252-253
 consequences, 253-255, 257-258
 contribution of new hires during
 transition, 251-252
 drag created by new hires on existing
 team, 252
 nomographs, 264-265
 non-linearity, 260-261
 productivity, 263-264
 spreadsheets, 265, 267
 suggestions for, 261-262
 guidelines, iteration estimating
 guidelines, 115
 Guisan, Olivier, 287

H

Hamming, Richard, 22
 Hawthorne Effect, 210
 heat death, 212
 Heisenberg’s Uncertainty Principle, 209-210
 Hello world, 71
 Heppenheimer, T. A., 20
 high-trust environments, 148, 153-154, 175
 Highsmith, James, 24
 honoring commitments, 174
 horseshoes, miscalibrations, 46
 human behavior, curves, 130-131
 human factor, 32

human nature, 270
 humility, 282
 humor, managing teams, 38
 Hybertsson, Henrik, 199

I

ICD (implanted cardioverter defibrillator), 3
 impedance mismatch, 163
 imprinting, 33
 inception, 137, 140
 increasing probability of success, 100-101
The Inferno, 276
 input mistakes, 18
 inquest, Vasa, 200
 inspiration, managing teams, 35
 instincts, trusting when managing teams, 39
 integrity, 272-273
 and software, 274
 iterations, 48, 114
 communication, 164-165
 estimating guidelines, 115
 phases, 137-139
 issues, avoiding, 244
 iterative development, 43-44, 118
 applied learning, 53-54
 build processes, 84
 project pyramid, 103
 rhythm, 135-137
 technology, 57-58
 versus waterfall development
 business implications, 54-55
 staffing effects, 55-57

J-K

Jacobsson, Hein, 199
 job-based model, compensation, 184-185
 job hunting, culture and values, 279-280

K&R, 70

Kant, Immanuel, 201

Katzenbach, Jon R., 34
 Kennedy, John F., 12
 Kernighan, Brian W., 70
 Kludge, 198
 koan, 192
 Krantz, Gene, 274
 Kroll, Per, 44
 Kruchten, Philippe, 44, 137, 222
 culture, 270
 KSLOC (kilo source lines of code), 113

L

labor content, 257
 labor cost, 260
 Laika, 12
 large project chicken, 175
 Larsen, Howard, 180
 Laws of Motion, analogies, 203-205
 leaders, 35, 147
 leadership
 continuity of, 199
 customer-focused culture, 276
 by example, 276
 pay for performance, 276
 learning
 new programming languages, 70-71
 standard problems, 71-74, 76
 Rational Unified Process, 138
 learning and completion curves, 140
 learning curves, 130
 Lencione, Patrick, 269
 Leroy, Pascal, 97, 220
 Leroy, Roscoe, 44-45
 calculating, 226-228, 230, 232-235, 239
 calendars, 110-111
 commitment, 168-169, 174-175
 deadlines, 171-172
 definition of, 170
 excuses, 170-173
 honoring, 174

- large project chicken, 175
- Texas handshake, 169-170
- communication, 158-159, 162-164
 - closure, 162
 - establishing ownership, 160-161
 - iteration, 164-165
 - not suggesting solutions, 161-162
 - remember who you're dealing with, 159-160
- estimating, 108-111, 114, 120
- long vectors, 48-49
- short vectors, 47
- square roots, 111
- life of software, 215-216
- life outside of work, managing teams, 38-39
- lifetime meal tickets, 14
- light, 213
 - speed of lights, 206
- limitations of project pyramid, 101-105
- listening, 7-8
 - managing teams, 36
- lognormal, 97
- lognormal distribution, 101, 104
- long vectors, 48-49
- lots of rules and no mercy, 82
- Lovitt, John, 192

M

- machers, 281, 284-286, 289
- malice, avoiding when managing teams, 37-38
- managers
 - audience for this book, 6
 - learning new programming languages, 70-71
 - standard problems, 71-74, 76
- managing teams, 34
 - anticipating obstacles, 35-36
 - avoiding malice, 37-38
 - focus, 34

- focus on facts, 37
- humor, 38
- inspiration, 35
- life outside of work, 38-39
- listening, 36
- stability, 37
- trust your instincts, 39
- mapping
 - politics, 152-153
 - team members, 191
- Marasco, Andrew, 20
- Maxwell, James Clerk, 211
- Mayo, Elton; Hawthorne Effect, 210
- measurements, 209
- mechanical drawing, 16
- menschs, 281, 286-288
- Meretsky, Wayne, 40
- mergers, 277-278
- milestones, 26-27
- miscalibrations, horseshoes, 46
- missile gap, 12
- mission-critical applications, 4
- mistakes, learning from others, 34
- modeling UML, 60-61
 - abstraction, 66
 - first example, 61
 - relevance to software, 65-66
 - second example, 61-63
 - third example, 63-65
- models
 - naïve model. *See* naïve model
 - non-linear relationships, 260-261
- momentum, 133
- Monday, 226
 - solution to fantasy baseball probabilities, 234-236, 238-239
- monitoring, 27
- Moore, Geoffrey, 131
- motion, Newton's Laws of Motion (analogies), 203-205
- mountain climbing, 23

multipliers, consequences of naïve model,
254-256

The Mythical Man-Month, 250

N

naïve model, 251

assumptions, 252-253

consequences of, 253-255, 257-258

adding useful hours to products, 256

cost, 257-259

nailing the multiplier, 254-256

contribution of new hires during
transition, 251-252

drag created by new hires on existing
team, 252

example of, 259-260

Nagasaki bomb, 12

negative payoff, 97

negotiating, 157

net force, 127

new efforts, continuity, 278-279

new hires, 250

contributions during transition, 251-252

effect on existing teams, 252

evaluating, 262

growth, 263

Newton, Sir Isaac, 127

Laws of Motion, analogies, 203-205

Newton's Second Law, 126-127

nomographs, 264-265

non-linear relationships, 260-261

non-linearity, 260-261

notation, abstraction, 66

O

observation, 7-8

OD (organizational development), 249

"off by a factor of ten," 17

Ohm's Law, 65

Ohms, 64

oil changes, 217

organizational politics, difficulty of build
processes, 80-81

organizing teams, 25-26

overpaid, 182

ownership, 160-161

P

parabolic position graphs, 130

Parkinson's Law, 53

partners, continuity, 278

paths in crisis, 246

paying for software upgrades, 218

batteries packaged with software,
218-223

people

machers, 284-286

menschs, 286-288

population distribution, 288-290

schleppers, 282-284

percent completion curve, 126

performance, 181-182

period of transition, 251

Perrow, Mike, 147

personality, 270

Peter Principle, 187

phases, 281

iterations, 137-139

machers, 284-286

menschs, 286-288

schleppers, 282-284

physics

acceleration of a vertical ball toss,
129-130

analogies, 201-203

electromagnetism, 211

entropy, 211

Newton's Laws of Motion, analogies,
203-205

- Netwon's Second Law, 126-127
 - quantum mechanics, 207-211
 - thermodynamics, 211-212
 - trajectory of a vertical ball toss, 129
 - velocity of a vertical ball toss, 128-129
- piracy, 223
- planck, 208-209
- planning
 - decisiveness, 28
 - goals, 29
 - handling risk, 27-28
 - milestones, 27
 - monitoring and recordkeeping, 27
 - scheduling, 26-27
 - scope, 24
- plateaus, 131
- PM (project management), 249
- political process, definition of, 147
- politics, 145, 150
 - bad politics, 152, 154-155
 - compromise, 148-149
 - context of, 146
 - good politics, 150-151
 - high-trust environments, 153-154
 - mapping, 152-153
 - neutral politics, 151
 - scenarios, 147-148
- population distribution, 288-290
- position, 127
- potential, 182
- precision, 109
- predictability, 123
 - scheduling, 119
- primacy of time, 90
- prioritizing risks, 52-53
- probabilities, 226, 228-229, 232-235, 239
 - generating more, 230
- probability of success, 94
 - project pyramid, 99-100
 - increasing, 100-101
- problem-solving, 21
 - problem-solving clock, 6-9
 - problems, fixing, 245
 - product versus prototype, 200
 - productivity. *See also* growth
 - flow, 181-182
 - growth, 263-264
 - new hires, 263
 - programming languages, managers learning
 - new languages, 70-71
 - standard problems, 71-74, 76
 - project completion graphs, 130
 - project management (PM), 249
 - project managers, adjusting schedules for, 122-123
 - project pyramid, 90-91, 93
 - altitude variable, 94
 - distribution, 97-99
 - iterative development, 103
 - limitations of, 101-105
 - probability of success, 99-101
 - risk, 92, 105
 - scaling, 95-97
 - volume, 94-95
 - project velocity curves, 132
 - projects
 - failure, common causes of, 30-31
 - human factor, 32
 - success of, 31-32
 - prototype versus product, 200
 - prototyping, 51
 - pseudoscientific jargon, 213
 - pyramid model, results of, 101
- Q-R**
 - quadratic algorithms, 172
 - quality, 92-93
 - of software, 4-5
 - quantum mechanics, analogies, 207-211
 - Queen Elizabeth I, 277
 - quick studies, 131

ramps, 130
 random number generators, 226
 random numbers, 228, 234. *See also*
 probabilities
 Raskin, Jef, 220
 rates of change, 127
 Rational Software, 5-6
 acquisition, 278
Rational Unified Process, 137
 reality and curves, 134-135
 recordkeeping, 27
 refreshing embedded software, 216-217
 registers, 272
 relativity, analogies, 205-207
 religion, culture, 271
 repeatable build process, 78
 resources, 92
 respect, 158
 results
 iterations and phases, 138-140
 values, 273-274
 rhythm
 build rhythm, 125
 curves of human behavior, 131
 iterataive development, 135-137
 risk, 27-28, 92
 prioritizing, 52-53
 project pyramid, 105
 risk targeting, 51-52
 Ritchie, Dennis M., 70
 Roebling, John, 199
 Royce, Walker, 44, 113, 137

S

S-curves, 130
 Sadler, Mark, 214
 sandbaggers, 120
 sandbagging, 176
 sandboxes, 79
 Sandstro, Anders, 198
 Sarbanes-Oxley Act of 2002, 276
 scaling, 250
 project pyramid, 95-97
 scarcity premiums, 183
 scenarios of politics, 147-148
 scheduling, 26-27, 117-118, 120-121
 adjusting schedules for different
 managers, 122-123
 calibrating, 120, 122-123
 commitments, 175-176
 estimates, 119
 final week, 123
 predictability, 119, 123
 Roscoe's graph, 122
 square root, 120
 "two-schedule" game, 117
 schleppers, 281-284, 289
 schrodinger, 208
 science, 13, 213
 physics. *See* physics
 scope, 92-93
 prelude to planning, 24
 scope creep, 127
 scope management, 90
 searching for a job, values and culture,
 279-280
 selecting teams, 25
 shipping the product, 77-78
 build processes, difficulty of, 79-84
 Shooman, Martin L., 251
 short vectors, 47, 49
 applied learning, 51
 software development, 50
 signing up. *See* commitment, 167
 skill levels, 180
 skills-based model, compensation, 183-184
 slide rules, 16-17
 slope, 127
 Smith, Douglas K., 34
 So What? Test, The Animal Game, 74-75

software
 importance of good software, 4-5
 life of, 215-216
 packaged with batteries, 218-223
 upgrading, 218
 packaging batteries with software, 218-223
 and values, 274-275
 software development managers, 6
 software development, short vectors, 50
 software development projects, developing, 96
 software piracy, 223
 solar power, 223
 sound, 206
 space race, 12
 Special Theory of Relativity, 207
 speed, 93, 129
 speed of light, 206
 speed of sound, 206
 spreadsheets, growth, 265-267
 Sputnik, 12
 Sputnik II, 12
 square roots
 estimating, 114
 Leroy, Roscoe, 111
 scheduling, 120
 stability, managing teams, 37
 staffing effects, waterfall development versus iterative development, 55-57
 standard problems, learning new programming languages, 71-74, 76
 starter dough, 278
 states. *See* phases
 stogies, 50
 strong cultures, 271, 280
 subscriptions, 218
 success
 definition of, 101-102
 of projects, ingredients for, 31-32

sunk cost, 215
 synthesizing, 8-9

T

tables
 Mapping Politics, 153
 Learning and Completion Metrics for Phases in the Rational Unified Process, 138
 Product of Total Force Peak Value and Time Interval Length, 141
 Results of Using the Pyramid Model and Lognormal Distribution, 101
 tangents, slope of, 127
 team members, mapping, 191
 teams
 competition, 29
 managing, 34
 anticipating obstacles, 35-36
 avoiding malice, 37-38
 focus, 34
 focus on facts, 37
 humor, 38
 inspiration, 35
 life outside of work, 38-39
 listening, 36
 stability, 37
 trust instincts, 39
 organizing, 25-26
 selecting, 25
 testing, 8-9, 200
 tetrahedron model, 92
 Texas handshake, 169-170
 Theory of Relativity, 207
 Thermodynamics, analogies, 211-212
 “throwing money at the problem,” 12
 time, 92
 tools, difficulty of build processes, 83-84
 trade-offs, project pyramid. *See* project pyramid

traditions, 276
 trajectory of a vertical ball toss, 128
 transition, 137, 140
 trusting instincts, managing teams, 39
 Turing's Machine, 213
 turning points, in crisis, 245-246
 "two-schedule" game, 117
 Tzu, Sun, 152

U

UCM (Unified Change Management), 82
 UML (Unified Modeling Language), 60-61
 abstraction, 66
 first example, 61
 relevance to software, 65-66
 second example, 61-63
 third example, 63-65
 undecideability, 213
 underpaid, 183
 update releases, 77
 upgrading
 devices, 217
 software, 218
 packaging software with batteries,
 218-223
 Urmi, Jaak, 197

V

values
 core values, 272
 corporate values, defining, 271-274
 customer focus, 273
 integrity, 272-273
 job hunting, 279-280
 results, 273-274
 and software, 274-275
 Vasa, 197-199
 design, 199

 inquest, 200
 leadership, 199
 product versus prototype, 200
 testing, 200
 Vasa Museum, 197
 vectors, 49
 long vectors, 48-49
 short vectors, 47, 49
 applied learning, 51
 software development, 50
 velocity, 127
 percent completion curves, 132
 of a vertical ball toss, 128
 Voltaire, 83
 volume, project pyramid, 94-95

W

Walker, John, 52
 waterfall approach, 46
 waterfall development versus iterative
 development
 business implications, 54-55
 staffing effects, 55-57
 weak cultures, 271
 Westheimer, F. H., 52
 Wideman, Max, 105
 iron triangle, 90
 Williams, Ted, 240
 win-win, compensation, 190
 Woods, Tiger, 122